Algorithm-Architecture Affinity - Parallelism Changes the Picture

Abildgren, Rasmus; Šaramentovas, Aleksandras; Ruzgys, Paulius; Koch, Peter; Le Moullec, Yannick

Publication date:
2007

Document Version
Publisher's PDF, also known as Version of record

Algorithm-Architecture Affinity – Parallelism Changes the Picture

Rasmus Abildgren*, Aleksandras Šaramentovas†, Paulius Ruzgys†, Peter Koch†, and Yannick Le Moullec†

*Center for Embedded Software Systems(CISS)
Aalborg University, Selma Lagerlöfs Vej 300,
DK-9220 Aalborg East, Denmark
rab@es.aau.dk

†Center for Software Define Radio (CSDR)
Aalborg University, Fredriks Bajers Vej 12,
DK-9220 Aalborg East, Denmark
{aleksara,paulius,pk,ylm}@es.aau.dk

***Abstract*—Reducing the time-to-market factor is a challenge for many embedded systems designers. In that respect, hardware-software partitioning is a key issue which has been studied during the last two decades. In this paper we present an extension to recent works dealing with metrics for guiding the hardware-software partitioning step. This extension builds upon and complement our own work with metrics in the Design Trotter project, and is combined with the affinity metric approach. We show that the proposed extension improves the original affinity metric in terms of parallelism detection, and thus can help system designers to make wiser hardware-software partitioning decisions, which in turn reduces the time-to-market factor.**

I. Introduction

In order to achieve more advanced and faster services in embedded systems, increasingly sophisticated algorithms are used. To keep abreast with the increased need for processing power, heterogeneous multiprocessor platforms are introduced, which includes GPPs, DSPs and FPGAs.

Introducing this variety of processing elements (PEs), not only increases the computational capacity of embedded systems but also adds various computational properties. To exploit this increased capacity and properties, the designer needs to find the best suited PEs for the different system functionalities. By considering these facts together with all the system constraints (Area, Time, Power, Price, Development Time), it becomes a non-trivial task to decide how the system functionality should be mapped on the architecture.

To handle this task system level design methodologies have been developed, including structured design space exploration (DSE). A suite of academic DSE frameworks, e.g. [1]–[3], as well as commercial tools have been proposed, in order to provide the design engineer with qualitative information for partitioning.

Exploring the design space with optimising for different constraints is known to be $\mathcal{NP}$ hard [4]. The DSE in these frameworks is therefore carried out as heuristic simulations, which still can be a time-consuming but necessary task for state-of-the-art large scale products. Large companies can usually find these resources and keep up with their competitors.

However, small and medium enterprises (SMEs), which typically sell state-of-the-art products of much smaller volumes, must also stay on the competitive edge. They are also restricted by the time-to-market factor, and can also benefit from using system level design methodologies (SLD) and tools. Unfortunately, many SMEs can not afford tools and specialists like big companies, and therefore have problems with changing their design methodology into SLD methodology.

We have examined the design methodology of a high-tech company in Denmark and found that the design space exploration phase in their overall design trajectory is limited in the sense that their partitioning depends on prior design, designers intuition and experience, and in rare cases on ad hoc analysis. Danish Technological Institute, a consulting company helping many SMEs incorporating new research results, agrees on that picture in most SMEs [5].

As a consequence of sticking to ad hoc design methodologies, SMEs development often run into situations where redesigning part of the system is necessary and therefore increases the time-to-market.

In this paper we propose an extension to the existing affinity metric proposed in [6] for guiding the partitioning of the system specification, and help making the DSE faster and easier. The rest of the paper is organised as follows. In section II, the existing affinity metric is presented and examples for the need of an extension to the original metric are shown. In section III the new proposed metric for parallelism is presented. The benefits of the proposed parallelism metric are illustrated in section IV by means of a Reed-Solomon decoder case-study. Finally we conclude in section V.

II. Affinity Metric

This section summarises the affinity metric proposed by D. Sciuto et.al. in [6], [7], and argues for the need of an extension of this metric. The affinity metric is designed to guide the design partitioning of system specification between general purpose processors, DSP processor, and FPGA/ASIC. The metric consists of a triplet of values $(A_{GPP}, A_{DSP}, A_{FPGA})$ indicating the match between the processing elements and the examined code. The individual values in the metric are calculated based on 14 other metrics which are designed to measure the source code for certain patterns highly correlated with architectural properties. The measurement is a static analysis of the source code and the metrics are defined as ratios between lines with specific properties, e.g., the ratio between lines with a condition and the total number of lines, or

defined as the number of assignment of a special type related to the total number of assignments. The metrics measure properties such as data types, Harvard architecture patterns, MAC patterns, and bit manipulation.

To illustrate how the affinity metric works on a real life example, we have applied it onto c-code (Fig 3) calculating a matrix multiplication. The results of the different metrics are shown in table I:

TABLE I
THE AFFINITY VALUE FOR THE MATRIX MULTIPLICATION ALGORITHM, WHERE A_{xxxx} INDICATES THE MATCH BETWEEN THE PROCESSING ELEMENT TYPE AND THE CODE. 0 =NO MATCHING, 1 =PERFECT MATCH.

A_{GPP}	A_{DSP}	A_{FPGA}
0.89	0.96	0.39

The normalised metric values indicate that the best architecture matching the algortihm is a DSP architecture, which the designer could easily rely on. An in-depth analysis of the code shows that besides the already extracted properties from the affinity metric, a high degree of inherent parallelism is present in the matrix multiplication algorithm. This is further discussed in section III. A high degree of inherent parallelism indicates that the algorithm is suited for parallel execution. This is one property of a FPGA architecture, and the original affinity metric does not consider it.

III. PARALLELISM METRIC

From the analysis of the matrix multiplication shown in Fig 3, we see that the inherent parallelism of an algorithm is an important parameter. Therefore it would be beneficial to measure the degree of inherent parallelism in the algorithm and use this in calculating the A_{FPGA} value of the affinity metric.

One of the first metrics considering the parallelism is Amdahl's speedup metric [8]. Here the potential execution speedup of an algorithm is defined as the ratio between the sequential execution, and the fully parallelised execution. What determines the fully parallelised execution is the critical path in the algorithm.

This is also the case for more recent parallelism metrics e.g. [9], [10], so let us consider the critical path by looking at precedence graphs.

Definition 1: Let $G = (N, E)$ represent the precedence graph of a method, m, where N represents the set of nodes n_i and E is the set of edges $e_{i,j}$. A node n_i can have a source node and a destination node. If the node does not have a source node, it is defined as a start node, and if the node does not have a destination, it is a sink node. If a dependency between two nodes; the parent node, n_i and the child node, n_j, exists, it is connected with an edge $e_{i,j}$. The node, n_j, cannot execute before it has obtained data from its parent(s).

Using definition 1, we can now express the critical path of algorithm using the following definition:

Definition 2: The critical path, CP, is a set of nodes $n_{start}, \ldots, n_i, \ldots, n_{sink}$ and associated edges $e_{start,h}, \ldots, e_{i,j}, \ldots, e_{k,sink}$ forming a path, p, from a start node, n_{start}, to a sink node, n_{sink}, for which the sum of costs are a maximum:

$$CP = \max cost(\{n_{start}, e_{start,h}, n_h, \ldots, n_i, e_{i,j}, n_j, \ldots \\ \ldots, n_k, e_{k,sink}, n_{sink}\}) \quad (1)$$

A way to measure the inherent parallelism that uses the critical path is the γ metric developed in our previous work [9] which is defined as:

$$\gamma = \frac{N}{CP} \quad (2)$$

where we consider the nodes to be atomic, meaning that N represents the total number of operations in the precedence graph.

The metric described in (2) expresses the level of inherent parallelism of the algorithm by calculating the ratio between the number of operations in the algorithm, and the number of operations in the critical path. In this case, where we consider all nodes as basic operations, N is equivalent with the total number of nodes N. This metric is organised such that with no inherent parallelism its gives the value 1. The metric value increases along with the inherent parallelism.

The affinity metric [7] on the other hand is in comparison a normalised measure, where zero indicates the worst match and one indicates a perfect match between the algorithm and the architectural property. Using the γ for expressing the inherent parallelism will lead to non-comparable results. A metric expressing the parallelism together with the affinity metric should have the same normalised properties. To suit these properties we can rewrite the γ metric into a normalised metric:

$$\gamma' = 1 - \frac{CP}{N} \quad (3)$$

The affinity metric is based on textural analysis of the source code and therefore does not refer to the number of operations, critical path or any of the terms used above for γ and γ'. Instead it operates with source lines which contain certain patterns.

In order to cope with the parallelism measure inside this source line based framework, we propose a new metric, θ, inspired by the γ' metric. θ is defined as:

$$\theta = 1 - \frac{S_{CP}}{S_m} \quad (4)$$

where S_{CP} is the number of source lines included in the critical path and S_m is the total number of source lines in the code. To emphasise the weight of the critical path, a loop unrolling is need to be performed before measuring S_m and S_{CP} of the θ metric.

This way of expressing the parallelism is not equivalent with γ' since every source line in a high level language will usually lead to more than one atomic operation. The danger is that the number of atomic operations highly depends on the programmers coding style. A compact code will result in

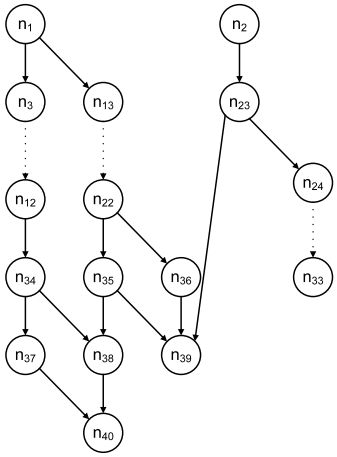

Fig. 1. Precedence graph of random 1 algorithm.

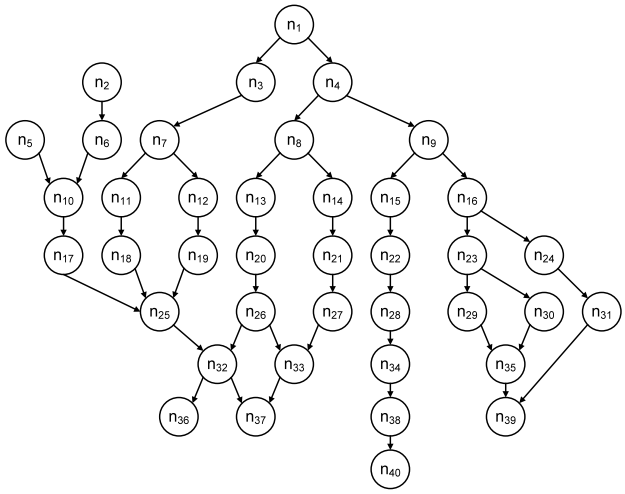

Fig. 2. Precedence graph of random 2 algorithm.

more operations per source line than a fragmented code with many intermediated/temporary variables which come close to one operation per code line. It is therefore impossible to obtain the same precision, as the modified and normalised γ' metric.

To examine their differences, extreme cases, i.e. a purely sequential and a fully parallel execution as well as two random cases have been considered. The two random execution graphs are shown in Fig 1 and Fig 2. Comparing the γ' metric and the θ metric on these cases provides us with the results shown in the four first lines of table II. We here consider $N = 40$ in the precedence graphs, where a source line on average corresponds to four nodes. The sequential execution gives, as expected, the same result for both metrics i.e., 0. The fully parallel execution however, gives a slightly different result for the two metrics, $\gamma' = 0.975$ and $\theta = 0.9$. None of them reach the value 1 for a full parallel execution, because of the way CP is defined. But we notice that θ gives a lower score than the γ' metric. This is due to the smaller number of code lines compared with the number of nodes, which influences the ratio. For the random case there are larger differences (0.65 vs. 0.56) and (0.7 vs. 0.75).

TABLE II
DIFFERENCES BETWEEN THE γ' AND θ METRIC.

	γ'	θ
Sequential:	0	0
Parallel:	0.975	0.9
Random 1	0.65	0.56
Random 2	0.7	0.75
Matrix Multiplication:	0.999	0.989

Even though the θ metric and the γ' metric do not give similar results, θ still gives a good indication of the algorithms affinity to a parallel architecture. Let us discuss this issue by re-considering the matrix multiplication case given by:

$$\mathbf{C} = \mathbf{AB} \tag{5}$$

```
int matrixMul(static int A[X*Y],
              static int B[Y*Z],
              static int C[X*Z])
{
  int *p_a = &A[0] ;
  int *p_b = &B[0] ;
  int *p_c = &C[0] ;

  int f ;
  int i ;
  int k ;

  for (k = 0 ; k < Z ; k++)
    {
      p_a = &A[0] ;  /* point to the beginning of array A */

      for (i = 0 ; i < X; i++)
         {
           p_b = &B[k*Y] ;    /* take next column */
           *p_c =  0 ;

           for (f = 0 ; f < Y; f++) /* do multiply */
              *p_c += *p_a++ * *p_b++ ;

           *p_c++ ;
         }
    }
  return(&C[0]) ;
}
```

Fig. 3. Matrix multiplication example.

where $\mathbf{C} \in \mathbb{R}^{X\times Z}$, $\mathbf{A} \in \mathbb{R}^{X\times Y}$, $\mathbf{B} \in \mathbb{R}^{Y\times Z}$ are matrixes where X, Y, Z denotes the dimensions. Here the dimensions are $X = Y = Z = 10$. The c-code taken from the DSPstone project [11] is shown in Fig 3, and we see that the kernel of the algorithm consists of multiplications, memory reads and writes together with some indexing controls. A precedence graph of the kernel of the algorithm is shown in Fig 4. The results of the examination of the algorithm with the two metrics are also shown in table II. From this we see that there is an insignificant difference between the two metrics (i.e., 0.999 and 0.989), which is due to the high number of nodes and unrolled source lines. From these cases it appears that the newly proposed metric θ serves its purpose of indicating parallelism.

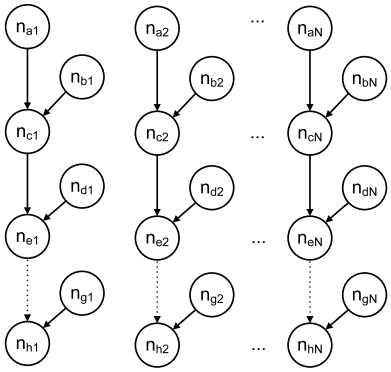

Fig. 4. Precedence graph of the kernel of the matrix multiplication example.

TABLE III
THE ORIGINAL AFFINITY METRIC VALUES FOR GPP, DSP, AND FPGA AND THE PROPOSED METRIC (FPGA&θ) FOR THE REED-SOLOMON DECODER ALGORITHM. THE PERFORMANCE (LATENCY) OF THE DIFFERENT ARCHITECTURES ARE ALSO SHOWN.

	GPP	DSP	$FPGA$	$FPGA\&\theta$
Affinity	0.717	0.795	0.205	0.806
Latency $[\mu s]$	-	514	2278	244

IV. CASE STUDY

In this section we present a case study, which expresses the benefits of the introduced metric, before selecting the architecture for a Reed-Solomon decoder.

A. Reed-Solomon Decoder

Reed-Solomon codes are a forward error correction codes used in many modern communication systems. The decoder is able to detect and correct some bit errors which have occurred doing the transmission. It is an algorithm which involves many conditional branches in order to detect and repair errors.

The algorithm has been examined with the affinity metric, and the results are shown in table III. The table shows the original affinity metric values for GPP, DSP and FPGA architectures and the affinity metric for FPGA with our new extension (added as an extra parameter for FPGA metric before normalisation as in [7]). We see that the Reed-Solomon decoder has the highest score (0.795) on a DSP architecture with the original affinity metric, however, the score for FPGA architecture increases significantly (from 0.205 to 0.806) when including our extension, and thereby gets the highest score. To verify the results, the algorithm has been implemented on a Analog Devices TigerSHARK ADSP-TS201 DSP and a Xilinx Virtex II FPGA, in high-level languages (C and Handel-C, respectively). The latency for decoding one block was measured on both platforms. The FPGA implementation was done in two steps: first, a version without exploiting the parallelism, which corresponds to the original affinity metric interpretation, and second, a version exploiting the inherent parallelism. These latencies are also shown in table III.

Inspecting the results shows that the best performance is obtained by the parallelised FPGA implementation, with a latency of $244\mu s$. We can then deduce that using the original affinity value for FPGA in this case will not disclose the architectures potential for the Reed-Solomon algorithm. Without considering the parallelism, the designer would make an inefficient partitioning choice.

Using the extended metric that we propose gives a better indication of the affinity between algorithm and FPGA architecture, thus helps the designer to make wiser partitioning decisions.

V. CONCLUSION

In this paper we have proposed an extention of the affinity metric [6], in order to improve the capability to measure the algorithm-architecture affinity for FPGA. The extension consists of a new metric derived from some of our previous work [9]. This new metric provides a mean for measuring the inherent parallelism of the algorithm inside the source code. We have shown that adding this new metric to the original affinity metric improves its score for FPGA matching.

REFERENCES

[1] C. Hylands, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, Y. Xiong, Y. Zhao, and H. Zheng, "Overview of the ptolemy project," Technical memorandum ucb/erl m03/25, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California 94720, July 2003.

[2] C. Erbas, A. D. Pimentel, M. Thompson, and S. Polstra, "A framework for system-level modeling and simulation of embedded systems architectures," *EURASIP Journal on Embedded Systems*, 2007.

[3] J. Riihimäki, P. Kukkala, T. Kangas, M. Hännikäinen, and T. D. Hämäläinen, "Interfacing uml 2.0 for multiprocessor system-on-chip design flow," in *Proceedings of International Symposium on System-on-Chip*, November 2005, pp. 108 – 111.

[4] Z. Á. Mann and A. Orbán, "Optimization problems in system-level synthesis," in *Proceedings of the 3rd Hungarian-Japanese Symposium on Discrete Mathematics and Its Applications*, 2003.

[5] T. S. Olesen, "Private conversation about Danish SMEs design methodologies," August 2006.

[6] C. Brandolese, W. Fornaciari, L. Pomante, F. Salice, and D. Sciuto, "Affinity-driven system design exploration for heterogeneous multiprocessor soc," *Computers, IEEE Transactions on*, vol. 55, no. 5, pp. 508–519, May 2006.

[7] D. Sciuto, F. Salice, L. Pomante, and W. Fornaciari, "Metrics for design space exploration of heterogeneous multiprocessor embedded systems," in *Hardware/Software Codesign, 2002. CODES 2002. Proceedings of the Tenth International Symposium on*, 6-8 May 2002, pp. 55–60.

[8] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *AFIPS spring joint computer conference*, 1967.

[9] Y. Le Moullec, N. Ben Amor, J-Ph. Diguet, M. Abid, and J-L. Philippe, "Multi-granularity metrics for the era of strongly personalized SOCs," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003.

[10] G. C. Sih and E. A. Lee, "A compile-time scheduling heuristic for internocnection-constrained heterogenous processor architectures," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 4, 1993.

[11] "DSP Stone," 1995, http://www.ert.rwth-aachen.de/Projekte/Tools/DSPSTONE/dspstone.html.